**Features of Python:**

Python is indeed an exciting and powerful language. Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

* **Simple:** Python is a simple and minimalistic language. This pseudo-code nature of Python is one of its greatest strengths. It allows you to concentrate on the solution to the problem rather than the syntax.
* **Easy to Learn:** Python has simple syntax.
* **Free and Open Source:** Python is an example of a FLOSS (Free/Libre and Open Source Software). In simple terms, you can freely distribute copies of this software, read the software's source code, make changes to it, use pieces of it in new free programs, and that you know you can do these things.
* **High-level Language:** When you write programs in Python, you never need to bother about low-level details such as managing the memory used by your program.
* **Portable:** Due to its open-source nature, Python has been ported to many platforms. All your Python programs will work on any of these platforms without requiring any changes at all. However, you must be careful enough to avoid any system-dependent features. You can use Python on Linux, Windows, Macintosh, Solaris, OS/2, Amiga, AROS, AS/400, BeOS, OS/390, z/OS, Palm OS, QNX, VMS, Psion, Acorn RISC OS, VxWorks, PlayStation, Sharp Zaurus, Windows CE and PocketPC !
* **Interpreted:** A program written in a compiled language like C or C++ is translated from the source language i.e. C/C++ into a language spoken by your computer (binary code i.e. 0s and 1s) using a compiler with various flags and options. When you run the program, the linker/loader software just stores the binary code in the computer's memory and starts executing from the first instruction in the program.

When you use an interpreted language like Python, there is no separate compilation and execution steps. You just run the program from the source code. Internally, Python converts the source code into an intermediate form called byte codes and then translates this into the native language of your specific computer and then runs it. All this makes using Python so much easier. You just run your programs - you never have to worry about linking and loading with libraries, etc.

* **Object Oriented:** Python supports procedure-oriented programming as well as object-oriented programming. In procedure-oriented languages, the program is built around procedures or functions which are nothing but reusable pieces of programs. In object-oriented languages, the program is built around objects which combine data and functionality. Python has a very powerful but simple way of doing object-oriented programming, especially, when compared to languages like C++ or Java.
* **Extensible:** If you need a critical piece of code to run very fast, you can achieve this by writing that piece of code in C, and then combine that with your Python program.
* **Embeddable:** You can embed Python within your C/C++ program to give scripting capabilities for your program's users.
* **Extensive Libraries:** The Python Standard Library is huge. It can help you do various things involving regular expressions, documentation generation, unit testing, threading, databases, web browsers, CGI, ftp, email, XML, XML-RPC, HTML, WAV files, cryptography, GUI(graphical user interfaces) using Tk, and also other system-dependent stuff. Remember, all this is always available wherever Python is installed. This is called the "batteries included" philosophy of Python.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below:

* It supports functional and structured programming methods as well as OOP.
* It can be used as a scripting language or can be compiled to byte-code for building large applications.
* It provides very high-level dynamic data types and supports dynamic type checking.
* IT supports automatic garbage collection.
* It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

**History and Future of Python**

Python was developed by Guido van Rossum in the late 1980s and early 1990s at the National Research Institute for Mathematics and Computer Science in the Netherlands. The implementation of Python was started in the December 1989.

In February 1991, van Rossum published the code (labeled version 0.9.0) to alt.sources.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL). Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

* In 1994, Python 1.0 was released with new features like: lambda, map, filter, and reduce.
* Python 2.0 added new features like: list comprehensions, garbage collection system.
* On December 3, 2008, Python 3.0 (also called "Py3K") was released. It was designed to rectify fundamental flaw of the language.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, UNIX shell and other scripting languages. ABC programming language is said to be the predecessor of Python language which was capable of Exception Handling and interfacing with Amoeba Operating System.

Python has a huge user base that is constantly growing. It is a stable language that is going to stay for long. Python has a bright future ahead of it supported by a huge community of OS developers. Python is a high speed dynamic language. Therefore it works well in applications like photo development and has been embedded in programs such as GIMP and Paint Shop Pro.

**The applications of Python are:**

Python is known for its general purpose nature that makes it applicable in almost each domain of software development. Python as a whole can be used in any sphere of development.

Here, we are specifying applications areas where python can be applied.

1. **Web Applications:** We can use Python to develop web applications. It provides libraries to handle internet protocols such as HTML and XML, JSON, Email processing, request, beautifulSoup, Feedparser etc. It also provides Frameworks such as Django, Pyramid, Flask etc to design and delelop web based applications. Some important developments are: PythonWikiEngines, Pocoo, PythonBlogSoftware etc.
2. **Desktop GUI Applications:** Python provides Tk GUI library to develop user interface in python based application. Other useful toolkits wxWidgets, Kivy, pyqt are useable on several platforms. The Kivy is popular for writing multi-touch applications.
3. **Software Development:** Python is helpful for software development process. It works as a support language and can be used for build control and management, testing etc.
4. **Scientific and Numeric:** Python is popular and widely used in scientific and numeric computing. Some useful library and package are SciPy, Pandas, IPython etc. SciPy is group of packages of engineering, science and mathematics.
5. **Business Applications:** Python is used to build Business applications like ERP and e-commerce systems. Python is a high level application platform.
6. **Console Based Application:** We can use Python to develop console based applications. For example: IPython.
7. **Audio or Video based Applications:** Python is awesome to perform multiple tasks and can be used to develop multimedia applications. Some of real applications are: TimPlayer, cplay etc.
8. **3D CAD Applications:** To create CAD application Fandango is a real application which provides full features of CAD.
9. **Enterprise Applications:** Python can be used to create applications which can be used within an Enterprise or an Organization. Some real time applications are: OpenErp, Tryton, Picalo etc.
10. **Applications for Images:** Using Python several application can be developed for image. Applications developed are: VPython, Gogh, imgSeek etc.

**Installing Python**

**Getting Python:**

The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python <https://www.python.org/>

You can download Python documentation from <https://www.python.org/doc/>. The documentation is available in HTML, PDF, and PostScript formats.

**Installing Python:**

Python distribution is available for a wide variety of platforms. You need to download only the binary code applicable for your platform and install Python. If the binary code for your platform is not available, you need a C compiler to compile the source code manually. Compiling the source code offers more flexibility in terms of choice of features that you require in your installation.

***Here is a quick overview of installing Python on various platforms:***

**UNIX and Linux Installation:**

Here are the simple steps to install Python on Unix/Linux machine.

* Open a Web browser and go to <https://www.python.org/downloads/>.
* Follow the link to download zipped source code available for Unix/Linux.
* Download and extract files.
* Editing the Modules/Setup file if you want to customize some options.
* run ./configure script
* make
* make install

This installs Python at standard location /usr/local/bin and its libraries at /usr/local/lib/pythonXX where XX is the version of Python.

**Windows Installation:**

Here are the steps to install Python on Windows machine.

* Open a Web browser and go to <https://www.python.org/downloads/>.
* Follow the link for the Windows installer python-XYZ.msi file where XYZ is the version you need to install.
* To use this installer python-XYZ.msi, the Windows system must support Microsoft Installer 2.0. Save the installer file to your local machine and then run it to find out if your machine supports MSI.
* Run the downloaded file. This brings up the Python install wizard, which is really easy to use. Just accept the default settings, wait until the install is finished, and you are done.
* Macintosh Installation
* Recent Macs come with Python installed, but it may be several years out of date. See [http://www.python.org/download/mac/](https://www.python.org/download/mac/) for instructions on getting the current version along with extra tools to support development on the Mac. For older Mac OS's before Mac OS X 10.3 (released in 2003), MacPython is available.

**Setting up PATH:**

Programs and other executable files can be in many directories, so operating systems provide a search path that lists the directories that the OS searches for executables. The path is stored in an environment variable, which is a named string maintained by the operating system. This variable contains information available to the command shell and other programs.

The path variable is named as PATH in UNIX or Path in Windows (UNIX is case sensitive; Windows is not).

In Mac OS, the installer handles the path details. To invoke the Python interpreter from any particular directory, you must add the Python directory to your path.

**Setting path at Unix/Linux:**

* To add the Python directory to the path for a particular session in Unix −
* In the csh shell − type setenv PATH "$PATH:/usr/local/bin/python" and press Enter.
* In the bash shell (Linux) − type export ATH="$PATH:/usr/local/bin/python" and press Enter.
* In the sh or ksh shell − type PATH="$PATH:/usr/local/bin/python" and press Enter.

Note − /usr/local/bin/python is the path of the Python directory

**Setting path at Windows:**

* To add the Python directory to the path for a particular session in Windows −
* At the command prompt − type path %path%;C:\Python and press Enter.

Note − C:\Python is the path of the Python directory

**Python Environment Variables**

Here are important environment variables, which can be recognized by Python −

|  |  |
| --- | --- |
| **S.No.** | **Variable & Description** |
| 1 | **PYTHONPATH**  It has a role similar to PATH. This variable tells the Python interpreter where to locate the module files imported into a program. It should include the Python source library directory and the directories containing Python source code. PYTHONPATH is sometimes preset by the Python installer. |
| 2 | **PYTHONSTARTUP**  It contains the path of an initialization file containing Python source code. It is executed every time you start the interpreter. It is named as .pythonrc.py in Unix and it contains commands that load utilities or modify PYTHONPATH. |
| 3 | **PYTHONCASEOK**  It is used in Windows to instruct Python to find the first case-insensitive match in an import statement. Set this variable to any value to activate it. |
| 4 | **PYTHONHOME**  It is an alternative module search path. It is usually embedded in the PYTHONSTARTUP or PYTHONPATH directories to make switching module libraries easy. |

**Running Python :** There are three different ways to start Python. They are:

1. **Interactive Interpreter:** You can start Python from Unix, DOS, or any other system that provides you a command-line interpreter or shell window.

Enter python the command line.

Start coding right away in the interactive interpreter.

$python # Unix/Linux

or

python% # Unix/Linux

or

C:> python # Windows/DOS

**Here is the list of all the available command line options:**

|  |  |
| --- | --- |
| **S.No.** | **Option & Description** |
| 1 | -d  It provides debug output. |
| 2 | -O  It generates optimized bytecode (resulting in .pyo files). |
| 3 | -S  Do not run import site to look for Python paths on startup. |
| 4 | -v  verbose output (detailed trace on import statements). |
| 5 | -X  disable class-based built-in exceptions (just use strings); obsolete starting with version 1.6. |
| 6 | -c cmd  run Python script sent in as cmd string |
| 7 | file  run Python script from given file |

1. **Script from the Command-line:** A Python script can be executed at command line by invoking the interpreter on your application, as in the following:

$python script.py # Unix/Linux

or

python% script.py # Unix/Linux

or

C: >python script.py # Windows/DOS

Note − Be sure the file permission mode allows execution.

1. **Integrated Development Environment**: You can run Python from a Graphical User Interface (GUI) environment as well, if you have a GUI application on your system that supports Python.

* Unix − IDLE is the very first Unix IDE for Python.
* Windows − PythonWin is the first Windows interface for Python and is an IDE with a GUI.
* Macintosh − The Macintosh version of Python along with the IDLE IDE is available from the main website, downloadable as either MacBinary or BinHex'd files.

If you are not able to set up the environment properly, then you can take help from your system admin. Make sure the Python environment is properly set up and working perfectly fine.

**Writing and Executing First Python Program**

Let us execute programs in different modes of programming:

1. **Interactive Mode Programming:**

Invoking the interpreter without passing a script file as a parameter brings up the following prompt:

$ python

Python 2.4.3 (#1, Nov 11 2010, 13:34:43)

[GCC 4.1.2 20080704 (Red Hat 4.1.2-48)] on linux2

Type "help", "copyright", "credits" or "license" for more information.

>>>

Type the following text at the Python prompt and press the Enter:

>>> print "Hello, Python!"

If you are running new version of Python, then you would need to use print statement with parenthesis as in print ("Hello, Python!"). However in Python version 2.4.3, this produces the following result:

Hello, Python!

1. **Script Mode Programming:**

Invoking the interpreter with a script parameter begins execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active.

Let us write a simple Python program in a script. Python files have extension .py. Type the following source code in a test.py file:

print "Hello, Python!"

We assume that you have Python interpreter set in PATH variable. Now, try to run this program as follows:

$ python test.py

This produces the following result:

Hello, Python!

Let us try another way to execute a Python script. Here is the modified test.py file −

#!/usr/bin/python

print "Hello, Python!"

We assume that you have Python interpreter available in /usr/bin directory. Now, try to run this program as follows:

$ chmod +x test.py # This is to make file executable

$./test.py

This produces the following result:

Hello, Python!

1. **Python Example using Interactive Shell:**

Python interactive shell is used to test the code immediately and does not require to write and save code in file.

Python code is simple and easy to run. Here is a simple Python code that will print "Welcome to Python".

A simple python example is given below.

>>> a="Welcome To Python"

>>> print a

Welcome To Python

>>>

Here we are using IDLE to write the Python code. Detail explanation to run code is given in Execute Python section.

A variable is defined named "a" which holds "Welcome To Python". "print" statement is used to print the content. Therefore "print a" statement will print the content of the variable. Therefore, the output "Welcome To Python" is produced.

**Python 3.4 Example:**

In python 3.4 version, you need to add parenthesis () in a string code to print it.

>>> a=("Welcome To Python Example")

>>> print a

Welcome To Python Example

>>>

**How to execute python:**

To execute Python code, we can use any approach.

1) **Interactive Mode:** Python provides Interactive Shell to execute code immediately and produce output instantly. To get into this shell, write python in the command prompt and start working with Python.

Press Enter key and the Command Prompt will appear like:

Now we can execute Python commands.

2) **Script Mode:** Using Script Mode, we can write Python code in a separate file of any editor in operating system. Save it by .py extension.

Now open Command prompt and execute.

NOTE: Path in the command prompt should be location of saved file where you have saved your file.

3) **Using IDE (Integrated Development Environment):** We can execute Python code using a Graphical User Interface (GUI).

Click on Start button -> All Programs -> Python -> IDLE(Python GUI)

We can use both Interactive as well as Script mode in IDE.

1) **Using Interactive mode:**

Execute our Python code on the Python prompt and it will display result simultaneously.

2) **Using Script Mode:**

i) Click on Start button -> All Programs -> Python -> IDLE(Python GUI)

ii) Python Shell will be opened. Now click on File -> New Window.

A new Editor will be opened. Write our Python code here.

Click on file -> save as

Run code by clicking on Run in the Menu bar.

Run -> Run Module

Result will be displayed on a new Python shell

**Python Identifiers**

A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (\_) followed by zero or more letters, underscores and digits (0 to 9).

Python does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive programming language. Thus, Manpower and manpower are two different identifiers in Python.

**Here are naming conventions for Python identifiers:**

* Class names start with an uppercase letter.
* All other identifiers start with a lowercase letter.
* Starting an identifier with a single leading underscore indicates that the identifier is private.
* Starting an identifier with two leading underscores indicates a strongly private identifier.
* If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

**Reserved Words**

The following list shows the Python keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| and | exec | not | elif | in | while |
| assert | finally | or | else | is | with |
| break | for | pass | except | lambda | yield |
| class | from | print | def | if | return |
| continue | global | raise | del | import | try |

**Lines and Indentation**

Python provides no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced.

The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. For example:

if True:

print "True"

else:

print "False"

However, the following block generates an error:

if True:

print "Answer"

print "True"

else:

print "Answer"

print "False"

Thus, in Python all the continuous lines indented with same number of spaces would form a block.

**Multi-Line Statements:**

Statements in Python typically end with a new line. Python does, however, allow the use of the line continuation character (\) to denote that the line should continue. For example:

total = item\_one + \

item\_two + \

item\_three

Statements contained within the [], {}, or () brackets do not need to use the line continuation character.

**For example:**

days = ['Monday', 'Tuesday', 'Wednesday',

'Thursday', 'Friday']

**Quotation in Python:**

Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.

The triple quotes are used to span the string across multiple lines. For example, all the following are legal:

word = 'word'

sentence = "This is a sentence."

paragraph = """This is a paragraph. It is

made up of multiple lines and sentences."""

**Comments in Python:**

A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.

#!/usr/bin/python

# First comment

print "Hello, Python!" # second comment

This produces the following result:

Hello, Python!

You can type a comment on the same line after a statement or expression:

name = "Madisetti" # This is again comment

You can comment multiple lines as follows:

# This is a comment.

# This is a comment, too.

# This is a comment, too.

**Using Blank Lines:**

A line containing only whitespace, possibly with a comment, is known as a blank line and Python totally ignores it. In an interactive interpreter session, you must enter an empty physical line to terminate a multiline statement.

**Waiting for the User:**

The following line of the program displays the prompt, the statement saying “Press the enter key to exit”, and waits for the user to take action:

#!/usr/bin/python

raw\_input("\n\nPress the enter key to exit.")

Here, "\n\n" is used to create two new lines before displaying the actual line. Once the user presses the key, the program ends. This is a nice trick to keep a console window open until the user is done with an application.

**Multiple Statements on a Single Line:**

The semicolon ( ; ) allows multiple statements on the single line given that neither statement starts a new code block.

Here is a sample snip using the semicolon:

import sys; x = 'foo'; sys.stdout.write(x + '\n')

**Multiple Statement Groups as Suites:**

A group of individual statements, which make a single code block are called suites in Python. Compound or complex statements, such as if, while, def, and class require a header line and a suite.

Header lines begin the statement (with the keyword) and terminate with a colon ( : ) and are followed by one or more lines which make up the suite.

For example:

if expression :

suite

elif expression :

suite

else :

suite

**Command Line Arguments:**

Many programs can be run to provide you with some basic information about how they should be run. Python enables you to do this with -h −

$ python -h

usage: python [option] ... [-c cmd | -m mod | file | -] [arg] ...

Options and arguments (and corresponding environment variables):

-c cmd : program passed in as string (terminates option list)

-d : debug output from parser (also PYTHONDEBUG=x)

-E : ignore environment variables (such as PYTHONPATH)

-h : print this help message and exit

[ etc. ]

**Python Variables**

Variable is a name which is used to refer memory location. Variables are also known as identifier and used to hold a value.

In Python, we don't need to specify the type of variable because Python is a type infer language and smart enough to get variable type.

Variable names can be a group of both letters and digits, but they have to begin with a letter or an underscore. It is recomended to use lowercase letters for variable name. Rahul and rahul both are two different variables.

Note - Variable name should not be a keyword.

**Declaring Variable and Assigning Values:**

Python does not bound us to declare variable before using in the application. It allows us to create variable at required time.

We don't need to declare explicitly variable in Python. When we assign any value to the variable that variable is declared automatically.

The equal (=) operator is used to assign value to a variable.

Eg:

a=10

b=ravi

c=20000.67

Output:

>>>

10

ravi

20000.67

>>>

**Multiple Assignments:**

Python allows us to assign a value to multiple variables in a single statement which is also known as multiple assignments.

We can apply multiple assignments in two ways either by assigning a single value to multiple variables or assigning multiple values to multiple variables. Let’s see given examples.

1. **Assigning single value to multiple variables:**

Eg:

x=y=z=50

print x

print y

print z

Output:

>>>

50

50

50

>>>

1. **Assigning multiple values to multiple variables:**

Eg:

a,b,c=5,10,15

print a

print b

print c

Output:

>>>

5

10

15

>>>

The values will be assigned in the order in which variables appears.

**Basic Fundamentals:**

The basic fundamentals of Python are :

i)Tokens and their types.

ii) Comments

a)**Tokens:**

Tokens can be defined as a punctuator mark, reserved words and each individual word in a statement. Token is the smallest unit inside the given program.

There are following tokens in Python:

Keywords.

Identifiers.

Literals.

Operators.

**Python Literals**

Literals can be defined as a data that is given in a variable or constant. Python support the following literals:

1. **String literals:** String literals can be formed by enclosing a text in the quotes. We can use both single as well as double quotes for a String.

Eg:

"Aman" , '12345'

**Types of Strings:**

There are two types of Strings supported in Python:

1. **Single line String**- Strings that are terminated within a single line are known as Single line Strings.

**Eg:**

>>> text1='hello'

1. **Multi line String**- A piece of text that is spread along multiple lines is known as Multiple line String.

There are two ways to create Multiline Strings:

1. Adding black slash at the end of each line.

Eg:

>>> text1='hello\

user'

>>> text1

'hellouser'

>>>

1. Using triple quotation marks:-

Eg:

>>> str2='''''welcome

to

SSSIT'''

>>> **print** str2

welcome

to

SSSIT

>>>

1. **Numeric literals:** Numeric Literals are immutable. Numeric literals can belong to following four different numerical types.

|  |  |  |  |
| --- | --- | --- | --- |
| Int(signed integers) | Long(long integers) | float(floating point) | Complex(complex) |
| Numbers( can be both positive and negative) with no fractional part.eg: 100 | Integers of unlimited size followed by lowercase or uppercase L eg: 87032845L | Real numbers with both integer and fractional part eg: -26.2 | In the form of a+bj where a forms the real part and b forms the imaginary part of complex number. eg: 3.14j |

1. **Boolean literals:** A Boolean literal can have any of the two values: True or False.
2. **Special literals:** Python contains one special literal i.e., None.

None is used to specify to that field that is not created. It is also used for end of lists in Python.

Eg:

>>> val1=10

>>> val2=None

>>> val1

10

>>> val2

>>> print val2

None

>>>

1. **Literal Collections:** Collections such as tuples, lists and Dictionary are used in Python.

**Python Operators**

Operators are particular symbols that are used to perform operations on operands. It returns result that can be used in application.

Eg:

4 + 5 = 9

Here 4 and 5 are Operands and (+), (=) signs are the operators. This expression produces the output 9.

**Types of Operators:**

Python supports the following operators

Arithmetic Operators.

Relational Operators.

Assignment Operators.

Logical Operators.

Membership Operators.

Identity Operators.

Bitwise Operators.

**Arithmetic Operators:** The following table contains the arithmetic operators that are used to perform arithmetic operations.

|  |  |
| --- | --- |
| Operators | Description |
| // | Perform Floor division(gives integer value after division) |
| + | To perform addition |
| - | To perform subtraction |
| \* | To perform multiplication |
| / | To perform division |
| % | To return remainder after division(Modulus) |
| \*\* | Perform exponent(raise to power) |

**Eg:**

>>> 10+20

30

>>> 20-10

10

>>> 10\*2

20

>>> 10/2

5

>>> 10%3

1

>>> 2\*\*3

8

>>> 10//3

3

>>>

**Relational Operators:** The following table contains the relational operators that are used to check relations.

|  |  |
| --- | --- |
| Operators | Description |
| < | Less than |
| > | Greater than |
| <= | Less than or equal to |
| >= | Greater than or equal to |
| == | Equal to |
| != | Not equal to |
| <> | Not equal to(similar to !=) |

**Eg:**

>>> 10<20

True

>>> 10>20

False

>>> 10<=10

True

>>> 20>=15

True

>>> 5==6

False

>>> 5!=6

True

>>> 10<>2

True

>>>

**Assignment Operators:** The following table contains the assignment operators that are used to assign values to the variables.

|  |  |
| --- | --- |
| Operators | Description |
| = | Assignment |
| /= | Divide and Assign |
| += | Add and assign |
| -= | Subtract and Assign |
| \*= | Multiply and assign |
| %= | Modulus and assign |
| \*\*= | Exponent and assign |
| //= | Floor division and assign |

**Eg:**

>>> c=10

>>> c

10

>>> c+=5

>>> c

15

>>> c-=5

>>> c

10

>>> c\*=2

>>> c

20

>>> c/=2

>>> c

10

>>> c%=3

>>> c

1

>>> c=5

>>> c\*\*=2

>>> c

25

>>> c//=2

>>> c

12

>>>

**Logical Operators:** The following table contains the arithmetic operators that are used to perform arithmetic operations.

|  |  |
| --- | --- |
| Operators | Description |
| and | Logical AND(When both conditions are true output will be true) |
| or | Logical OR (If any one condition is true output will be true) |
| not | Logical NOT(Compliment the condition i.e., reverse) |

**Eg:**

a=5>4 and 3>2

print a

b=5>4 or 3<2

print b

c=not(5>4)

print c

Output:

>>>

True

True

False

>>>

**Membership Operators:** The following table contains the membership operators.

|  |  |
| --- | --- |
| Operators | Description |
| in | Returns true if a variable is in sequence of another variable, else false. |
| not in | Returns true if a variable is not in sequence of another variable, else false. |

**Eg:**

a=10

b=20

list=[10,20,30,40,50];

if (a in list):

    print "a is in given list"

else:

    print "a is not in given list"

if(b not in list):

    print "b is not given in list"

else:

    print "b is given in list"

Output:

>>>

a is in given list

b is given in list

>>>

**Identity Operators:** The following table contains the identity operators.

|  |  |
| --- | --- |
| Operators | Description |
| is | Returns true if identity of two operands are same, else false |
| is not | Returns true if identity of two operands are not same, else false. |

**Eg:**

a=20

b=20

if( a is b):

    print  a,b have same identity

else:

    print a, b are different

b=10

if( a is not b):

    print  a,b have different identity

else:

    print a,b have same identity

Output

>>>

a,b have same identity

a,b have different identity

>>>

**PYTHON STRINGS**

Python string is a built-in type text sequence. It is used to handle textual data in python. Python Strings are immutable sequences of Unicode points. Creating Strings are simplest and easy to use in Python.

We can simply create Python String by enclosing a text in single as well as double quotes. Python treat both single and double quotes statements same.

**Accessing Python Strings:**

In Python, Strings are stored as individual characters in a contiguous memory location. The benefit of using String is that it can be accessed from both the directions (forward and backward).

Both forward as well as backward indexing are provided using Strings in Python. Forward indexing starts with 0,1,2,3 ...and Backward indexing starts with -1,-2,-3,-4,....

**Eg:**

str[0]='P'=str[-6] , str[1]='Y' = str[-5]  ,  str[2] = 'T' = str[-4]  ,  str[3] = 'H' = str[-3]

str[4] = 'O' = str[-2]  ,  str[5] = 'N' = str[-1].

**Python String Example:** Here, we are creating a simple program to retrieve String in reverse as well as normal form.

name="Rajat"

length=len(name)

i=0

for n in range(-1,(-length-1),-1):

    print name[i],"\t",name[n]

    i+=1

Output:

>>>

R t

a a

j j

a a

t R

>>>

**Python Strings Operators:**

To perform operation on string, Python provides basically 3 types of Operators that are given below.

1. Basic Operators.
2. Membership Operators.
3. Relational Operators.

**Python String Basic Operators**

There are two types of basic operators in String "+" and "\*".

1. **String Concatenation Operator (+):**

The concatenation operator (+) concatenates two Strings and creates a new String.

Python String Concatenation Example:

>>> "ratan" + "jaiswal"

Output:

'ratanjaiswal'

>>>

|  |  |
| --- | --- |
| Expression | Output |
| '10' + '20' | '1020' |
| "s" + "007" | 's007' |
| 'abcd123' + 'xyz4' | 'abcd123xyz4' |

NOTE: Both the operands passed for concatenation must be of same type, else it will show an error.

Eg:

'abc' + 3

>>>

Output:

Traceback (most recent call last):

File "", line 1, in

'abc' + 3

TypeError: cannot concatenate 'str' and 'int' objects

>>>

1. **Python String Replication Operator (\*):**

Replication operator uses two parameters for operation, one is the integer value and the other one is the String argument.

The Replication operator is used to repeat a string number of times. The string will be repeated the number of times which is given by the integer value.

Python String Replication Example:

>>> 5\*"Vimal"

Output:

'VimalVimalVimalVimalVimal'

|  |  |
| --- | --- |
| Expression | Output |
| "soono"\*2 | 'soonosoono' |
| 3\*'1' | '111' |
| '$'\*5 | '$$$$$' |

NOTE: We can use Replication operator in any way i.e., int \* string or string \* int. Both the parameters passed cannot be of same type.

**Python String Membership Operators**

Membership Operators are already discussed in the Operators section. Let see with context of String.

There are two types of Membership operators

1. in:"in" operator returns true if a character or the entire substring is present in the specified string, otherwise false.
2. not in:"not in" operator returns true if a character or entire substring does not exist in the specified string, otherwise false.

Python String membership operator Example:

>>> str1="javatpoint"

>>> str2='sssit'

>>> str3="seomount"

>>> str4='java'

>>> st5="it"

>>> str6="seo"

>>> str4 in str1

True

>>> str5 in str2

>>> st5 in str2

True

>>> str6 in str3

True

>>> str4 not in str1

False

>>> str1 not in str4

True

**Python Relational Operators**

All the comparison (relational) operators i.e., (<,><=,>=,==,!=,<>) are also applicable for strings.

The Strings are compared based on the ASCII value or Unicode(i.e., dictionary Order).

Python Relational Operators Example:

>>> "RAJAT"=="RAJAT"

True

>>> "afsha">='Afsha'

True

>>> "Z"<>"z"

True

The ASCII value of a is 97, b is 98, c is 99 and so on. The ASCII value of A is 65,B is 66,C is 67 and so on. The comparison between strings are done on the basis on ASCII value.

**Python String Slice Notation**

Python String slice can be defined as a substring which is the part of the string. Therefore further substring can be obtained from a string. There can be many forms to slice a string, as string can be accessed or indexed from both the direction and hence string can also be sliced from both the directions.

**Python String Slice Syntax:**

<string\_name>[startIndex:endIndex],

<string\_name>[:endIndex],

<string\_name>[startIndex:]

**Python String Slice Example 1:**

>>> str="Nikhil"

>>> str[0:6]

'Nikhil'

>>> str[0:3]

'Nik'

>>> str[2:5]

'khi'

>>> str[:6]

'Nikhil'

>>> str[3:]

'hil'

**Note:** startIndex in String slice is inclusive whereas endIndex is exclusive.

String slice can also be used with Concatenation operator to get whole string.

**Python String Slice Example 2:**

>>> str="Mahesh"

>>> str[:6]+str[6:]

'Mahesh'

//here 6 is the length of the string.

**Python String Functions and Methods:**

Python provides various predefined or built-in string functions. They are as follows:

|  |  |
| --- | --- |
| capitalize() | It capitalizes the first character of the String. |
| count(string,begin,end) | It Counts number of times substring occurs in a String between begin and end index. |
| endswith(suffix ,begin=0,end=n) | It returns a Boolean value if the string terminates with given suffix between begin and end. |
| find(substring ,beginIndex, endIndex) | It returns the index value of the string where substring is found between begin index and end index. |
| index(subsring, beginIndex, endIndex) | It throws an exception if string is not found and works same as find() method. |
| isalnum() | It returns True if characters in the string are alphanumeric i.e., alphabets or numbers and there is at least 1 character. Otherwise it returns False. |
| isalpha() | It returns True when all the characters are alphabets and there is at least one character, otherwise False. |
| isdigit() | It returns True if all the characters are digit and there is at least one character, otherwise False. |
| islower() | It returns True if the characters of a string are in lower case, otherwise False. |
| isupper() | It returns False if characters of a string are in Upper case, otherwise False. |
| isspace() | It returns True if the characters of a string are whitespace, otherwise false. |
| len(string) | It returns the length of a string. |
| lower() | It converts all the characters of a string to Lower case. |
| upper() | It converts all the characters of a string to Upper Case. |
| startswith(str ,begin=0,end=n) | It returns a Boolean value if the string starts with given str between begin and end. |
| swapcase() | It inverts case of all characters in a string. |
| lstrip() | It removes all leading whitespace of a string and can also be used to remove particular character from leading. |
| rstrip() | It removes all trailing whitespace of a string and can also be used to remove particular character from trailing. |

**Python String capitalize() Method Example:**

This method capitalizes the first character of the String.

>>> 'abc'.capitalize()

Output:

'Abc'

**Python String count(string) Method Example:**

This method counts number of times substring occurs in a String between begin and end index.

msg = "welcome to sssit";

substr1 = "o";

print  msg.count(substr1, 4, 16)

substr2 = "t";

print  msg.count(substr2)

Output:

>>>

2

2

>>>

**Python String endswith(string) Method Example:**

This method returns a Boolean value if the string terminates with given suffix between begin and end.

string1="Welcome to SSSIT";

substring1="SSSIT";

substring2="to";

substring3="of";

print string1.endswith(substring1);

print string1.endswith(substring2,2,16);

print string1.endswith(substring3,2,19);

print string1.endswith(substring3);

Output:

>>>

True

False

False

False

>>>

**Python String find(string) Method Example:**

This method returns the index value of the string where substring is found between begin index and end index.

str="Welcome to SSSIT";

substr1="come";

substr2="to";

print str.find(substr1);

print str.find(substr2);

print str.find(substr1,3,10);

print str.find(substr2,19);

Output:

>>>

3

8

3

-1

>>>

**Python String index() Method Example:**

This method returns the index value of the string where substring is found between begin index and end index.

str="Welcome to world of SSSIT";

substr1="come";

substr2="of";

print str.index(substr1);

print str.index(substr2);

print str.index(substr1,3,10);

print str.index(substr2,19);

Output:

>>>

3

17

3

Traceback (most recent call last):

File "C:/Python27/fin.py", line 7, in

print str.index(substr2,19);

ValueError: substring not found

>>>

**Python String isalnum() Method Example:**

This method returns True if characters in the string are alphanumeric i.e., alphabets or numbers and there is at least 1 character. Otherwise it returns False.

str="Welcome to sssit";

    print str.isalnum();

str1="Python47";

print str1.isalnum();

Output:

>>>

False

True

>>>

**Python String isalpha() Method Example:**

It returns True when all the characters are alphabets and there is at least one character, otherwise False.

string1="HelloPython";    # Even space is not allowed

print string1.isalpha();

string2="This is Python2.7.4"

print string2.isalpha();

Output:

>>>

True

False

>>>

**Python String isdigit() Method Example:**

This method returns True if all the characters are digit and there is at least one character, otherwise False.

string1="HelloPython";

print string1.isdigit();

string2="98564738"

print string2.isdigit();

Output:

>>>

False

True

>>>

**Python String islower() Method Example:**

This method returns True if the characters of a string are in lower case, otherwise False.

string1="Hello Python";

print string1.islower();

string2="welcome to "

print string2.islower();

Output:

>>>

False

True

>>>

**Python String isupper() Method Example:**

This method returns False if characters of a string are in Upper case, otherwise False.

string1="Hello Python";

print string1.isupper();

string2="WELCOME TO"

print string2.isupper();

Output:

>>>

False

True

>>>

**Python String isspace() Method Example:**

This method returns True if the characters of a string are whitespace, otherwise false.

string1="    ";

print string1.isspace();

string2="WELCOME TO WORLD OF PYT"

print string2.isspace();

Output:

>>>

True

False

>>>

**Python String len(string) Method Example:**

This method returns the length of a string.

string1="    ";

print len(string1);

string2="WELCOME TO SSSIT"

print len(string2);

Output:

>>>

4

16

>>>

**Python String lower() Method Example:**

It converts all the characters of a string to Lower case.

string1="Hello Python";

print string1.lower();

string2="WELCOME TO SSSIT"

print string2.lower();

Output:

>>>

hello python

welcome to sssit

>>>

**Python String upper() Method Example:**

This method converts all the characters of a string to upper case.

string1="Hello Python";

print string1.upper();

string2="welcome to SSSIT"

print string2.upper();

Output:

>>>

HELLO PYTHON

WELCOME TO SSSIT

>>>

**Python String startswith(string) Method Example:**

This method returns a Boolean value if the string starts with given str between begin and end.

string1="Hello Python";

print string1.startswith('Hello');

string2="welcome to SSSIT"

print string2.startswith('come',3,7);

Output:

>>>

True

True

>>>

**Python String swapcase() Method Example:**

It inverts case of all characters in a string.

string1="Hello Python";

print string1.swapcase();

string2="welcome to SSSIT"

print string2.swapcase();

Output:

>>>

hELLO pYTHON

WELCOME TO sssit

>>>

**Python String lstrip() Method Example:**

It removes all leading whitespace of a string and can also be used to remove particular character from leading.

string1="    Hello Python";

print string1.lstrip();

string2="@@@@@@@@welcome to SSSIT"

print string2.lstrip('@');

Output:

>>>

Hello Python

welcome to world to SSSIT

>>>

**Python String rstrip() Method Example:**

It removes all trailing whitespace of a string and can also be used to remove particular character from trailing.

string1="    Hello Python     ";

print string1.rstrip();

string2="@welcome to SSSIT!!!"

print string2.rstrip('!');

Output:

>>>

Hello Python

@welcome to SSSIT

>>>

**Python List**

Python list is a data structure which is used to store various types of data. Lists are mutable i.e., modifiable. Python will not create a new list if we modify an element of the list.

The values stored in List are separated by commas (,) and enclosed within a square brackets ([]). We can store different type of data in a List and elements are stored in the index basis with starting index 0.

Value stored in a List can be retrieved using the slice operator ([] and [:]). The plus sign (+) is the list concatenation and asterisk (\*) is the repetition operator.

**Python List Syntax:**

<list\_name>=[value1,value2,value3,...,valuen];

It works as a container that holds other objects in a given order. We can perform various operations like insertion and deletion on list.

**Eg:**

data1=[1,2,3,4]

data2=['x','y','z']

data3=[12.5,11.6]

data4=['raman','rahul']

data5=[]

data6=['abhinav',10,56.4,'a']

**Syntax to Access Python List**

<list\_name>[index]

Python allows us to access value from the list by various ways.

**Eg:**

data1=[1,2,3,4];

data2=['x','y','z'];

print data1[0]

print data1[0:2]

print data2[-3:-1]

print data1[0:]

print data2[:2]

Output:

>>>

>>>

1

[1, 2]

['x', 'y']

[1, 2, 3, 4]

['x', 'y']

>>>

**Another Eg:**

>>> list=['aman',678,20.4,'saurav']

>>> list1=[456,'rahul']

>>> list

['aman', 678, 20.4, 'saurav']

>>> list[1:3]

[678, 20.4]

>>> list+list1

['aman', 678, 20.4, 'saurav', 456, 'rahul']

>>> list1\*2

[456, 'rahul', 456, 'rahul']

>>>

**Elements in Lists:**

Following are the pictorial representation of a list. We can see that it allows to access elements from both end (forward and backward).

Data=[1,2,3,4,5];

Data[0]=1=Data[-5] ,Data[1]=2=Data[-4],Data[2]=3=Data[-3],Data[3]=4=Data[-2],Data[4]=5=Data[-1].

**Note:** Internal Memory Organization

List do not store the elements directly at the index. In fact a reference is stored at each index which subsequently refers to the object stored somewhere in the memory. This is due to the fact that some objects may be large enough than other objects and hence they are stored at some other memory location.

**Python List Operations**

Apart from creating and accessing elements from the list, Python allows us to perform various other operations on the list. Some common operations are given below:

1. **Adding Python Lists:** In Python, lists can be added by using the concatenation operator(+) to join two lists.

**Example 1:**

list1=[10,20]

    list2=[30,40]

    list3=list1+list2

    print list3

Output:

>>>

    [10, 20, 30, 40]

    >>>

**Note:** '+'operator implies that both the operands passed must be list else error will be shown.

**Example 2:**

  list1=[10,20]

list1+30

print list1

Output:

Traceback (most recent call last):

        File "C:/Python27/lis.py", line 2, in <module>

            list1+30

1. **Python Replicating lists:** Replicating means repeating, It can be performed by using '\*' operator by a specific number of time.

**Example:**

list1=[10,20]

print list1\*1

Output:

>>>

[10, 20]

>>>

1. **Python List Slicing:** A subpart of a list can be retrieved on the basis of index. This subpart is known as list slice. This feature allows us to get sub-list of specified start and end index.

**Example:**

list1=[1,2,4,5,7]

print list1[0:2]

print list1[4]

list1[1]=9

print list1

Output:

>>>

[1, 2]

7

[1, 9, 4, 5, 7]

>>>

**Note:** If the index provided in the list slice is outside the list, then it raises an IndexError exception.

1. **Python List Other Operations**: Apart from above operations various other functions can also be performed on List such as Updating, Appending and Deleting elements from a List.
2. **Python Updating List:** To update or change the value of particular index of a list, assign the value to that particular index of the List.

**Example:**

data1=[5,10,15,20,25]

print "Values of list are: "

print data1

data1[2]="Multiple of 5"

print "Values of list are: "

print data1

Output:

>>>

Values of list are:

[5, 10, 15, 20, 25]

Values of list are:

[5, 10, 'Multiple of 5', 20, 25]

1. **Appending Python List:** Python provides, append() method which is used to append i.e., add an element at the end of the existing elements.

**Example:**

list1=[10,"rahul",'z']

print "Elements of List are: "

print list1

list1.append(10.45)

print "List after appending: "

print list1

Output:

>>>

Elements of List are:

[10, 'rahul', 'z']

List after appending:

[10, 'rahul', 'z', 10.45]

>>>

1. **Deleting Elements:** In Python, del statement can be used to delete an element from the list. It can also be used to delete all items from startIndex to endIndex.

**Example:**

list1=[10,'rahul',50.8,'a',20,30]

print list1

del list1[0]

print list1

del list1[0:3]

print list1

Output:

>>>

[10, 'rahul', 50.8, 'a', 20, 30]

['rahul', 50.8, 'a', 20, 30]

[20, 30]

>>>

**Python lists Method:**

Python provides various Built-in functions and methods for Lists that we can apply on the list. Following are the common list functions.

|  |  |
| --- | --- |
| Function | Description |
| min(list) | It returns the minimum value from the list given. |
| max(list) | It returns the largest value from the given list. |
| len(list) | It returns number of elements in a list. |
| cmp(list1,list2) | It compares the two list. |
| list(sequence) | It takes sequence types and converts them to lists. |

**Python List min() method Example:**

This method is used to get min value from the list.

list1=[101,981,'abcd','xyz','m']

list2=['aman','shekhar',100.45,98.2]

print "Minimum value in List1: ",min(list1)

print "Minimum value in List2: ",min(list2)

Output:

>>>

Minimum value in List1:  101

Minimum value in List2:  98.2

>>>

**Python List max() method Example:**

This method is used to get max value from the list.

list1=[101,981,'abcd','xyz','m']

list2=['aman','shekhar',100.45,98.2]

print "Maximum value in List : ",max(list1)

print "Maximum value in List : ",max(list2)

Output:

>>>

Maximum value in List :  xyz

Maximum value in List :  shekhar

>>>

**Python List len() method Example:**

This method is used to get length of the the list.

list1=[101,981,'abcd','xyz','m']

list2=['aman','shekhar',100.45,98.2]

print "No. of elements in List1: ",len(list1)

print "No. of elements in List2: ",len(list2)

Output:

>>>

No. of elements in List1 :  5

No. of elements in List2 :  4

>>>

**Python List cmp() method Example:**

If elements are of the same type, perform the comparison and return the result. If elements are different types, check whether they are numbers. If numbers, perform comparison. If either element is a number, then the other element is returned. Otherwise, types are sorted alphabetically.

If we reached the end of one of the lists, the longer list is "larger." If both lists are same it returns 0.

**Example:**

list1=[101,981,'abcd','xyz','m']

list2=['aman','shekhar',100.45,98.2]

list3=[101,981,'abcd','xyz','m']

print cmp(list1,list2)

print cmp(list2,list1)

print cmp(list3,list1)

Output:

>>>

-1

1

0

>>>

**Python List list(sequence) method Example:**

This method is used to form a list from the given sequence of elements.

seq=(145,"abcd",'a')

data=list(seq)

print "List formed is : ",data

Output:

>>>

List formed is :  [145, 'abcd', 'a']

>>>

There are following built-in methods of List

|  |  |
| --- | --- |
| Methods | Description |
| index(object) | It returns the index value of the object. |
| count(object) | It returns the number of times an object is repeated in list. |
| pop()/pop(index) | It returns the last object or the specified indexed object. It removes the popped object. |
| insert(index,object) | It inserts an object at the given index. |
| extend(sequence) | It adds the sequence to existing list. |
| remove(object) | It removes the object from the given List. |
| reverse() | It reverses the position of all the elements of a list. |
| sort() | It is used to sort the elements of the List. |

**Python List index() Method Example:**

data = [786,'abc','a',123.5]

print "Index of 123.5:", data.index(123.5)

print "Index of a is", data.index('a')

Output:

>>>

Index of 123.5 : 3

Index of a is 2

>>>

**Python List count(object) Method Example:**

data = [786,'abc','a',123.5,786,'rahul','b',786]

print "Number of times 123.5 occured is", data.count(123.5)

print "Number of times 786 occured is", data.count(786)

Output:

>>>

Number of times 123.5 occured is 1

Number of times 786 occured is 3

>>>

**Python List pop()/pop(int) Method Example:**

data = [786,'abc','a',123.5,786]

print "Last element is", data.pop()

print "2nd position element:", data.pop(1)

print data

Output:

>>>

Last element is 786

2nd position element:abc

[786, 'a', 123.5]

>>>

**Python List insert(index,object) Method Example:**

data=['abc',123,10.5,'a']

data.insert(2,'hello')

print data

Output:

>>>

['abc', 123, 'hello', 10.5, 'a']

>>>

**Python List extend(sequence) Method Example:**

data1=['abc',123,10.5,'a']

data2=['ram',541]

data1.extend(data2)

print data1

print data2

Output:

>>>

['abc', 123, 10.5, 'a', 'ram', 541]

['ram', 541]

>>>

**Python List remove(object) Method Example:**

data1=['abc',123,10.5,'a','xyz']

data2=['ram',541]

print data1

data1.remove('xyz')

print data1

print data2

data2.remove('ram')

print data2

Output:

>>>

['abc', 123, 10.5, 'a', 'xyz']

['abc', 123, 10.5, 'a']

['ram', 541]

[541]

>>>

**Python List reverse() Method Example:**

list1=[10,20,30,40,50]

list1.reverse()

print list1

Output:

>>>

[50, 40, 30, 20, 10]

>>>

**Python List sort() Method Example:**

list1=[10,50,13,'rahul','aakash']

list1.sort()

print list1

Output:

>>>

[10, 13, 50, 'aakash', 'rahul']

>>>

**Python Tuples**

Tuple is another form of collection where different type of data can be stored. A tuple is a sequence of immutable objects, therefore tuple cannot be changed. It can be used to collect different types of object.

It is similar to list where data is separated by commas. Only the difference is that list uses square bracket and tuple uses parenthesis.

**Eg:**

>>> tuple=('rahul',100,60.4,'deepak')

>>> tuple1=('sanjay',10)

>>> tuple

('rahul', 100, 60.4, 'deepak')

>>> tuple[2:]

(60.4, 'deepak')

>>> tuple1[0]

'sanjay'

>>> tuple+tuple1

('rahul', 100, 60.4, 'deepak', 'sanjay', 10)

>>>

**Another Eg:**

>>> data=(10,20,'ram',56.8)

>>> data2="a",10,20.9

>>> data

(10, 20, 'ram', 56.8)

>>> data2

('a', 10, 20.9)

>>>

**NOTE:** If Parenthesis is not given with a sequence, it is by default treated as Tuple. There can be an empty Tuple also which contains no object.

**Example of empty tuple:**

tuple1=()

**Python Single Object Tuple Example:**

For a single valued tuple, there must be a comma at the end of the value.

Tuple1=(10,)

**Python Tuple of Tuples Example:**

Tuples can also be nested, it means we can pass tuple as an element to create a new tuple. See, the following example in which we have created a tuple that contains tuples the object.

tupl1='a','mahesh',10.56

tupl2=tupl1,(10,20,30)

print tupl1

print tupl2

Output:

>>>

('a', 'mahesh', 10.56)

(('a', 'mahesh', 10.56), (10, 20, 30))

>>>

**Accessing Tuple:**

Accessing of tuple is prity easy, we can access tuple in the same way as List.

Eg:

data1=(1,2,3,4)

data2=('x','y','z')

print data1[0]

print data1[0:2]

print data2[-3:-1]

print data1[0:]

print data2[:2]

Output:

>>>

1

(1, 2)

('x', 'y')

(1, 2, 3, 4)

('x', 'y')

>>>

**Elements in a Tuple:**

Data=(1,2,3,4,5,10,19,17)

Data[0]=1=Data[-8] , Data[1]=2=Data[-7] , Data[2]=3=Data[-6] ,

    Data[3]=4=Data[-5] , Data[4]=5=Data[-4] , Data[5]=10=Data[-3],

Data[6]=19=Data[-2],Data[7]=17=Data[-1]

**Python Tuple Operations:**

Python allows us to perform various operations on the tuple.

**Adding Tuples Example:**

Tuple can be added by using the concatenation operator(+) to join two tuples.

data1=(1,2,3,4)

data2=('x','y','z')

data3=data1+data2

print data1

print data2

print data3

Output:

>>>

(1, 2, 3, 4)

('x', 'y', 'z')

(1, 2, 3, 4, 'x', 'y', 'z')

>>>

**Note:** The new sequence formed is a new Tuple.

**Replicating Tuple Example:**

Replicating means repeating. It can be performed by using '\*' operator by a specific number of time.

tuple1=(10,20,30);

tuple2=(40,50,60);

print tuple1\*2

print tuple2\*3

Output:

>>>

(10, 20, 30, 10, 20, 30)

(40, 50, 60, 40, 50, 60, 40, 50, 60)

>>>

**Python Tuple Slicing Example:**

A subpart of a tuple can be retrieved on the basis of index. This subpart is known as tuple slice.

data1=(1,2,4,5,7)

print data1[0:2]

print data1[4]

print data1[:-1]

print data1[-5:]

print data1

Output:

>>>

(1, 2)

7

(1, 2, 4, 5)

(1, 2, 4, 5, 7)

(1, 2, 4, 5, 7)

>>>

**Note:** If the index provided in the Tuple slice is outside the list, then it raises an IndexError exception.

**Updating elements in a List:**

Elements of the Tuple cannot be updated. This is due to the fact that Tuples are immutable. Whereas the Tuple can be used to form a new Tuple.

**Eg:**

data=(10,20,30)

data[0]=100

print data

Output:

>>>

Traceback (most recent call last):

File "C:/Python27/t.py", line 2, in

data[0]=100

TypeError: 'tuple' object does not support item assignment

>>>

**Creating Tuple from Existing Example:**

We can create a new tuple by assigning the existing tuple, see the following example.

data1=(10,20,30)

data2=(40,50,60)

data3=data1+data2

print data3

Output:

>>>

(10, 20, 30, 40, 50, 60)

>>>

**Python Tuple Deleting Example:**

Deleting individual element from a tuple is not supported. However the whole of the tuple can be deleted using the del statement.

data=(10,20,'rahul',40.6,'z')

print data

del data      #will delete the tuple data

print data  #will show an error since tuple data is already deleted

Output:

>>>

(10, 20, 'rahul', 40.6, 'z')

Traceback (most recent call last):

File "C:/Python27/t.py", line 4, in

print data

NameError: name 'data' is not defined

>>>

**Functions of Tuple**

There are following in-built Type Functions

|  |  |
| --- | --- |
| Function | Description |
| min(tuple) | It returns the minimum value from a tuple. |
| max(tuple) | It returns the maximum value from the tuple. |
| len(tuple) | It gives the length of a tuple |
| cmp(tuple1,tuple2) | It compares the two Tuples. |
| tuple(sequence) | It converts the sequence into tuple. |

**Python Tuple min(tuple) Method Example:**

This method is used to get min value from the sequence of tuple.

data=(10,20,'rahul',40.6,'z')

print min(data)

Output:

>>>

10

>>>

**Python Tuple max(tuple) Method Example:**

This method is used to get max value from the sequence of tuple.

data=(10,20,'rahul',40.6,'z')

print max(data)

Output:

>>>

z

>>>

**Python Tuple len(tuple) Method Example:**

This method is used to get length of the tuple.

data=(10,20,'rahul',40.6,'z')

print len(data)

Output:

>>>

5

>>>

**Python Tuple cmp(tuple1,tuple2) Method Example:**

This method is used to compare tuples. If elements are of the same type, perform the comparison and return the result. If elements are different types, check whether they are numbers. If numbers, perform comparison. If either element is a number, then the other element is returned. Otherwise, types are sorted alphabetically .

If we reached the end of one of the lists, the longer list is "larger." If both list are same it returns 0.

data1=(10,20,'rahul',40.6,'z')

data2=(20,30,'sachin',50.2)

print cmp(data1,data2)

print cmp(data2,data1)

data3=(20,30,'sachin',50.2)

print cmp(data2,data3)

Output:

>>>

-1

1

0

>>>

**Advantages of Tuples:**

* Processing of Tuples are faster than Lists.
* It makes the data safe as Tuples are immutable and hence cannot be changed.
* Tuples are used for String formatting.

**Python Dictionary**

Dictionary is a collection which works on a key-value pair. It works like an associated array where no two keys can be same. Dictionaries are enclosed by curly braces ({}) and values can be retrieved by square bracket([]).

**Eg:**

>>> dictionary={'name':'charlie','id':100,'dept':'it'}

>>> dictionary

{'dept': 'it', 'name': 'charlie', 'id': 100}

>>> dictionary.keys()

['dept', 'name', 'id']

>>> dictionary.values()

['it', 'charlie', 100]

>>>

Dictionary is an unordered set of key and value pair. It is a container that contains data, enclosed within curly braces. The pair i.e., key and value is known as item. The key passed in the item must be unique.

The key and the value is separated by a colon(:). This pair is known as item. Items are separated from each other by a comma(,). Different items are enclosed within a curly brace and this forms Dictionary.

**Eg:**

data={100:'Ravi' ,101:'Vijay' ,102:'Rahul'}

print data

Output:

>>>

{100: 'Ravi', 101: 'Vijay', 102: 'Rahul'}

>>>

Dictionary is mutable i.e., value can be updated. Key must be unique and immutable. Value is accessed by key. Value can be updated while key cannot be changed. Dictionary is known as Associative array since the Key works as Index and they are decided by the user.

**Eg:**

plant={}

plant[1]='Ravi'

plant[2]='Manoj'

plant['name']='Hari'

plant[4]='Om'

print plant[2]

print plant['name']

print plant[1]

print plant

Output:

>>>

Manoj

Hari

Ravi

{1: 'Ravi', 2: 'Manoj', 4: 'Om', 'name': 'Hari'}

>>>

**Accessing Dictionary Values:**

Since Index is not defined, a Dictionary values can be accessed by their keys only. It means, to access dictionary elements we need to pass key, associated to the value.

**Syntax:**

<dictionary\_name>[key]

</dictionary\_name>

**Example:**

data1={'Id':100, 'Name':'Suresh', 'Profession':'Developer'}

data2={'Id':101, 'Name':'Ramesh', 'Profession':'Trainer'}

print "Id of 1st employer is",data1['Id']

print "Id of 2nd employer is",data2['Id']

print "Name of 1st employer:",data1['Name']

print "Profession of 2nd employer:",data2['Profession']

Output:

>>>

Id of 1st employer is 100

Id of 2nd employer is 101

Name of 1st employer is Suresh

Profession of 2nd employer is Trainer

>>>

**Updating Python Dictionary Elements:**

The item i.e., key-value pair can be updated. Updating means new item can be added. The values can be modified.

**Example:**

data1={'Id':100, 'Name':'Suresh', 'Profession':'Developer'}

data2={'Id':101, 'Name':'Ramesh', 'Profession':'Trainer'}

data1['Profession']='Manager'

data2['Salary']=20000

data1['Salary']=15000

print data1

print data2

Output:

>>>

{'Salary': 15000, 'Profession': 'Manager','Id': 100, 'Name': 'Suresh'}

{'Salary': 20000, 'Profession': 'Trainer', 'Id': 101, 'Name': 'Ramesh'}

>>>

**Deleting Python Dictionary Elements Example:**

del statement is used for performing deletion operation. An item can be deleted from a dictionary using the key only.

**Syntax:**

del  <dictionary\_name>[key]

</dictionary\_name>

Whole of the dictionary can also be deleted using the del statement.

**Example:**

data={100:'Ram', 101:'Suraj', 102:'Alok'}

del data[102]

print data

del data

print data   #will show an error since dictionary is deleted.

Output:

>>>

{100: 'Ram', 101: 'Suraj'}

Traceback (most recent call last):

File "C:/Python27/dict.py", line 5, in

print data

NameError: name 'data' is not defined

>>>

**Python Dictionary Functions and Methods:**

Python Dictionary supports the following Functions

|  |  |
| --- | --- |
| Functions | Description |
| len(dictionary) | It returns number of items in a dictionary. |
| cmp(dictionary1,dictionary2) | It compares the two dictionaries. |
| str(dictionary) | It gives the string representation of a string. |

Python Dictionary Methods

|  |  |
| --- | --- |
| Methods | Description |
| keys() | It returns all the keys element of a dictionary. |
| values() | It returns all the values element of a dictionary. |
| items() | It returns all the items (key-value pair) of a dictionary. |
| update(dictionary2) | It is used to add items of dictionary2 to first dictionary. |
| clear() | It is used to remove all items of a dictionary. It returns an empty dictionary. |
| fromkeys(sequence,value1)/ fromkeys(sequence) | It is used to create a new dictionary from the sequence where sequence elements form the key and all keys share the values? value1?. In case value1 is not give, it set the values of keys to be none. |
| copy() | It returns an ordered copy of the data. |
| has\_key(key) | It returns a boolean value. True in case if key is present in the dictionary, else false. |
| get(key) | It returns the value of the given key. If key is not present it returns none. |

**Python Dictionary len(dictionary) Example:**

It returns length of the dictionary.

data={100:'Ram', 101:'Suraj', 102:'Alok'}

print data

print len(data)

Output:

>>>

{100: 'Ram', 101: 'Suraj', 102: 'Alok'}

3

>>>

**Python Dictionary cmp(dictionary1,dictionary2) Example:**

The comparison is done on the basis of key and value.

If, dictionary1 == dictionary2, returns 0.

      dictionary1 < dictionary2, returns -1.

     dictionary1 > dictionary2, returns 1.

data1={100:'Ram', 101:'Suraj', 102:'Alok'}

data2={103:'abc', 104:'xyz', 105:'mno'}

data3={'Id':10, 'First':'Aman','Second':'Sharma'}

data4={100:'Ram', 101:'Suraj', 102:'Alok'}

print cmp(data1,data2)

print cmp(data1,data4)

print cmp(data3,data2)

Output:

>>>

-1

0

1

>>>

**Python Dictionary str(dictionary) Example:**

This method returns string formation of the value.

data1={100:'Ram', 101:'Suraj', 102:'Alok'}

print str(data1)

Output:

>>>

{100: 'Ram', 101: 'Suraj', 102: 'Alok'}

>>>

**Python Dictionary keys() Method Example:**

This method returns all the keys element of a dictionary.

data1={100:'Ram', 101:'Suraj', 102:'Alok'}

print data1.keys()

Output:

>>>

[100, 101, 102]

>>>

**Python Dictionary values() Method Example:**

This method returns all the values element of a dictionary.

data1={100:'Ram', 101:'Suraj', 102:'Alok'}

print data1.values()

Output:

>>>

['Ram', 'Suraj', 'Alok']

>>>

**Python Dictionary items() Method Example:**

This method returns all the items(key-value pair) of a dictionary.

data1={100:'Ram', 101:'Suraj', 102:'Alok'}

print data1.items()

Output:

>>>

[(100, 'Ram'), (101, 'Suraj'), (102, 'Alok')]

>>>

**Python Dictionary update(dictionary2) Method Example:**

This method is used to add items of dictionary2 to first dictionary.

data1={100:'Ram', 101:'Suraj', 102:'Alok'}

data2={103:'Sanjay'}

data1.update(data2)

print data1

print data2

Output:

>>>

{100: 'Ram', 101: 'Suraj', 102: 'Alok', 103: 'Sanjay'}

{103: 'Sanjay'}

>>>

**Python Dictionary clear() Method Example:**

It returns an ordered copy of the data.

data1={100:'Ram', 101:'Suraj', 102:'Alok'}

print data1

data1.clear()

print data1

Output:

>>>

{100: 'Ram', 101: 'Suraj', 102: 'Alok'}

{}

>>>

**Python Dictionary fromkeys(sequence)/ fromkeys(seq,value) Method Example:**

This method is used to create a new dictionary from the sequence where sequence elements forms the key and all keys share the values ?value1?. In case value1 is not give, it set the values of keys to be none.

sequence=('Id' , 'Number' , 'Email')

data={}

data1={}

data=data.fromkeys(sequence)

print data

data1=data1.fromkeys(sequence,100)

print data1

Output:

>>>

{'Email': None, 'Id': None, 'Number': None}

{'Email': 100, 'Id': 100, 'Number': 100}

**Python Dictionary copy() Method Example:**

This method returns an ordered copy of the data.

data={'Id':100 , 'Name':'Aakash' , 'Age':23}

data1=data.copy()

print data1

Output:

>>>

{'Age': 23, 'Id': 100, 'Name': 'Aakash'}

>>>

**Python Dictionary has\_key(key) Method Example:**

It returns a boolean value. True in case if key is present in the dictionary, else false.

data={'Id':100 , 'Name':'Aakash' , 'Age':23}

print data.has\_key('Age')

print data.has\_key('Email')

Output:

>>>

True

False

>>>

**Python Dictionary get(key) Method Example:**

This method returns the value of the given key. If key is not present it returns none.

data={'Id':100 , 'Name':'Aakash' , 'Age':23}

print data.get('Age')

print data.get('Email')

Output:

>>>

23

None

>>>

**Python 2.x Vs Python 3.x**

Important differences between Python 2.x and Python 3.x with examples

* Division operator
* print function
* Unicode
* xrange
* Error Handling
* \_future\_ module

**Division operator**

If we are porting code or executing the python 3.x code in python 2.x, it can be dangerous if integer division changes go unnoticed (since it doesn’t raise any error). It is preferred to use the floating value (like 7.0/5 or 7/5.0) to get the expected result when porting our code.

|  |
| --- |
| print 7 / 5  print -7 / 5    Output in Python 2.x  1  -2  Output in Python 3.x :  1.4  -1.4 |
|  |
|  |
|  |

**Print function**

This is the most well known change. In this the **print** function in Python 2.x is replaced by print() function in Python 3.x,i.e, to print in Python 3.x an extra pair of parenthesis is required.

|  |
| --- |
| print 'Hello, Geeks'      # Python 3.x doesn't support  print('Hope You like these facts')    Output in Python 2.x :  Hello, Geeks  Hope You like these facts    Output in Python 3.x :  File "a.py", line 1      print 'Hello, Geeks'                         ^  SyntaxError: invalid syntax |

As we can see, if we use parenthesis in python 2.x then there is no issue but if we don’t use parenthesis in python 3.x, we get SyntaxError.

**Unicode:**

In Python 2, implicit str type is ASCII but in Python 3.x implicit str type is Unicode.

|  |
| --- |
| print(type('default string '))  print(type(b'string with b '))    Output in Python 2.x (Bytes is same as str)  <type 'str'>  <type 'str'>    Output in Python 3.x (Bytes and str are different)  <class 'str'>  <class 'bytes'> |

Python 2.x also supports Unicode.

|  |
| --- |
| print(type('default string '))  print(type(u'string with b '))    Output in Python 2.x (Unicode and str are different)  <type 'str'>  <type 'unicode'>    Output in Python 3.x (Unicode and str are same)  <class 'str'>  <class 'str'> |

**xrange:**

xrange() of Python 2.x doesn’t exist in Python 3.x. In Python 2.x, range returns a list i.e. range(3) returns [0, 1, 2] while xrange returns a xrange object i.e., xrange(3) returns iterator object which work similar to Java iterator and generates number when needed. If we need to iterate over the same sequence multiple times, we prefer range() as range provides a static list. xrange() reconstructs the sequence every time. xrange() doesn’t support slices and other list methods. The advantage of xrange() is, it saves memory when task is to iterate over a large range.

In Python 3.x, the range function now does what xrange does in Python 2.x, so to keep our code portable, we might want to stick to using range instead. So Python 3.x’s range function is xrange from Python 2.x.

|  |
| --- |
| for x in xrange(1, 5):      print(x),    for x in range(1, 5):      print(x),  Output in Python 2.x  1 2 3 4 1 2 3 4    Output in Python 3.x  NameError: name 'xrange' is not defined |

**Error Handling:**

There is a small change in error handling in both versions. In Python 3.x, ‘as’ keyword is required.

|  |
| --- |
| try:      trying\_to\_check\_error  except NameError, err:      print err, 'Error Caused'   # Would not work in Python 3.x  Output in Python 2.x:  name 'trying\_to\_check\_error' is not defined Error Caused    Output in Python 3.x :  File "a.py", line 3      except NameError, err:                      ^  SyntaxError: invalid syntax  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| try:       trying\_to\_check\_error  except NameError as err: # 'as' is needed in Python 3.x       print (err, 'Error Caused')  Output in Python 2.x:  (NameError("name 'trying\_to\_check\_error' is not defined",), 'Error Caused')    Output in Python 3.x :  name 'trying\_to\_check\_error' is not defined Error Caused |

**future\_module:**

|  |
| --- |
| # In below python 2.x code, division works  # same as Python 3.x because we use  \_\_future\_\_  from \_\_future\_\_ import division   print 7 / 5  print -7 / 5 |

the idea of future module is to help in migration. We can use Python 3.x. if we are planning Python 3.x support in our 2.x code, we can ise\_future\_imports it in our code. For example, in below Python 2.x code, we use Python 3.x’s integer division behavior using future module.

Output:

1.4

-1.4